Testing

- Basic ideas and principles
- Traditional testing strategies
  - Unit, integration, regression, validation, system
- OO Testing techniques
- Application of traditional techniques to OO software

Motivation

- People are not perfect
  - We make errors in design and code
- Goal of testing: given some code, uncover as many errors are possible
- Important and expensive activity
  - Not unusual to spend 30-40% of total project effort on testing
  - For critical systems (e.g., flight control): cost can be several times the cost of all other activities combined

What is the goal of testing?

- Testing - process of executing a program to find errors
  - Successful test causes the program to FAIL
  - Can't expect goal to be demonstration that program is error-free
  - Testing can validate not verify!

A Way of Thinking

- Design and coding are creative
- Testing is destructive
  - The primary goal is to "break" the software
- Very often the same person does both coding and testing
  - Need "split personality": when you start testing, become paranoid and malicious
  - Surprisingly hard to do: people don't like finding out that they made mistakes

Testing Objectives

- Testing is a process of executing software with the intent of finding errors
- Good testing has a high probability of finding as-yet-undiscovered errors
- Successful testing discovers unknown errors
  - If did not find any errors, need to ask whether our testing approach is good

Basic Definitions

- Test case specifies
  - Inputs + pre-test state of the software
  - Expected results (outputs and state)
- Black-box testing: ignores the internal logic of the software
  - Given this input, was the output correct?
- White-box testing: uses knowledge of the internal structure of the software
  - e.g. write tests to "cover" internal paths

But Testing All Paths is impossible

- Assume we have 100 LOC in C
  - 2 nested loops (from 1-20 times each)
  - W/I inner loop have 4 if-then-else constructs which yields 10^{14} posisble paths
    - If we assume a processor can develop a test case, execute it and evaluate it in 1 millisecond, then working 24/7, the processor would take 3170 YEARS to test this program

Testing Principles

- Test must have an expected output
- Programmer should not test their own code or it becomes proofreading
- Need to thoroughly inspect all test results
- Test cases needed for the invalid and unexpected values as well as for the expected and valid conditions
- Must look to see if a program fails to do what it is supposed to do, and if it does what it is not supposed to do

Testing Principles

- Save tests for a regression suite
- Don't plan testing assuming no errors will be found
- Probability of more errors in a section of code is proportional to the number of errors already found in that section
  - Errors cluster -- very unintuitive, but observable
- Good testing is creatively and intellectually challenging

Traditional Testing Strategies

- Unit testing: per component
  - Can be done for multiple components in parallel
  - What's tested?
    - Interface, local data structures, boundary conditions, independent paths, error handling paths
  - Which errors found?
    - Computation problems, loop errors, bad error messages
  - May require test driver to run component under test (CUT) and stubs for components it calls
    - Stubs provide minimal functionality
    - Drivers and stubs are overhead to test process

Traditional Testing Strategies

- Integration testing:
  - Systematically combining components while checking for interface errors
  - Big Bang vs Incremental
  - Top down
    - Move down through control hierarchy of modules; can do depth-first or breadth-first
    - Replace each stub by a component as testing progresses downwards
    - Difficult if processing in lower components necessary to test upper components - makes stubs complicated to build

Traditional Testing Strategies

  - Bottom up
    - Begin testing with modules at bottom of hierarchy and work way upwards
      - Avoids need for stubs
      - Gather lower components into clusters which can be tested by a driver
- Regression testing
  - Save representative selection of tests to make sure functionality is not changed when code is changed

Traditional Testing Strategies

- Validation testing
  - Checks software behaves as user expects; conformity to requirements
  - May involve alpha and/or beta test releases
- System testing
  - To fully exercise the system to fully check integration of all elements (including HW platform)
  - Types: recovery testing, security testing, stress testing, performance testing

OO Testing Techniques

- Unit testing
  - Class testing
    - Check the state behavior and operations encapsulated in the class
    - Can apply usual white-box testing techniques to methods in the class
- Integration testing
  - Thread-based testing
    - Integrate set of classes needed to respond to one input or event for the system
    - Use regression tests to make sure no side effects are introduced
  - Use-based testing
    - Construct system by testing classes that use very few, if any server classes; continue to test classes dependent upon them, as go up the "uses" hierarchy layers

OO Testing Techniques

- Drivers may simulate the UI or to test groups of classes; Stubs may be necessary if not all classes are written
- Cluster testing
  - To find problems in collaborations of bad interactions or improper specifications

Applicability of Traditional Techniques

- Fault-based testing
  - Faults looked for during integration include bad operation calls or message connections
    - Unexpected result, wrong message used, incorrect invocation
    - Look for faults in the caller, not the callee
- Test cases and Inheritance
    - Subclass methods need to be tested independently from superclass methods they override
    - Inherited superclass methods may need to be retested when calling overriding subclass methods (for subclass objects)

Applicability of Traditional Techniques

- Scenario-based testing
  - Based on use cases
  - Often more complex than fault-based tests, because they exercise multiple subsystems
- Testing surface structure
  - Corresponds to black-box testing; observable structure by end-users
- Testing deep structure
  - Corresponds to white-box testing which 'covers' the control-flow or data-flow structure of the program

How to know when to stop testing?

- Decide based on some test-case design methodology
  - Only can use for some test phases
- When detect some pre-defined number of errors
  - Can use predictive models for estimation
- Examine number of errors found per unit of time
  - Decide if need to continue based on slope of graph
- In reality -- RUN OUT OF TIME